基于项编码的分布式频繁项集挖掘算法 *

郑静益，邓晓衡†

(中南大学 软件学院，长沙 410075)

摘　要：Apriori 算法是解决频繁项集挖掘最常用的算法之一，但多轮迭代扫描完整数据集的计算方式，严重影响算法效率且难以并行化处理。随着数据规模的持续增大，这一问题日益严重。针对这一问题，提出了一种基于项编码和 Spark 计算框架的 Apriori 并行化处理方法——IEBDA 算法，利用项编码完整保存项集信息，在不重复扫描完整数据集的情况下完成频繁项集挖掘，同时利用 Spark 的广播变量实现并行化处理。与其他分布式 Apriori 算法在不同规模的数据集上进行性能比较，发现 IEBDA 算法从第一轮迭代后加速效果明显。结果表明，该算法可以提高大数据环境下的多轮迭代的频繁项集挖掘效率。

关键词：频繁项集挖掘；Apriori 算法；大数据；分布式计算

中图分类号：TP391　　doi: 10.3969/j.issn.1001-3695.2017.11.0791

Novel distributed itemset mining algorithm based on item encoding

Zheng Jingyi, Deng Xiaoheng†

(*College of Software*, *Central South University*, *Changsha* 410075, *China*)

Abstract: Apriori is one of the most widely used algorithm to discover frequent patterns. However, scanning the entire dataset in each iteration makes this algorithm inefficient and hard to be in parallel. With the size of datasets gets larger continuously, this problem is becoming more and more serious. Therefore, a novel algorithm called IEBDA is proposed. This algorithm is a kind of parallelization of Apriori based on item encoding and Spark framework. Saving information of each itemset by item encoding so that it can finish frequent itemset mining without scanning the whole dataset repeatedly. The broadcast variables of Spark enables this algorithm to be in parallel. Compared with other distributed Apriori algorithms on datasets with different sizes, the acceleration of mining after the first iteration is obvious. The results show that this algorithm do efficiently improve the multi-iteratively frequent itemset mining in big data environment.

Key words: frequent itemset mining; Apriori algorithm; big data; distributed computation

0　引言

关联规则挖掘是基于规则的学习技术，致力于发现数据集中数据对象之间的重要关系。创建所有可能的项集规则需要大量的内存和 cpu 资源。因此，为了减少潜在项集的数量，只考虑利用频繁项集来建立关联规则。Apriori 是众多频繁项集挖掘算法中最常用且最易理解的算法之一。Apriori 算法[1-2]以迭代方式搜索 k 阶候选项集的频繁项集，其中 k 表示迭代的次数。通过预先设置好的支持度阈值，判断项集是否符合频繁条件。第 k 次迭代产生的频繁项集用于在第 k+1 轮迭代中找到更高阶频繁项集。

但是，随着大数据时代的来临，Apriori 算法存在缺陷被不断放大。首先，每次迭代中扫描整个数据集，数据规模的不断增大，使得算法计算消耗的时间急剧增加。其次，每轮迭代产生的候选项集数量也随着数据量的增加而增加，内存消耗问题无法忽视，严重时甚至会因为内存溢出而无法得出挖掘结果。Hadoop 和 Spark 等大数据平台的出现为 Apriori 算法的并行化处理提供了思路，然而 Hadoop 提供的 Map-Reduce 编程模型需要反复读写磁盘，并不适合于处理迭代式的算法，因此很多基于 Hadoop 改进的并行化 Apriori 算法和变种算法[3-11]，通过集群的形式，增加 cpu 和内存，能应用于大数据集并计算出最后结果，但是算法的效率仍然不尽如人意。而 Spark 基于内存的计算模型则可能是一种更好的解决思路。目前已经有不少基于 Spark 的 Apriori 算法优化工作[12-16]，相较之前的基于 Hadoop 的改进，取得了更好的效果，但是在中间候选项集的存储方式上仍有进一步改进的空间，因此在数据集的重复扫描问题上和候选项集剪枝工作上，仍然有进一步优化的可能。

本文的主要贡献如下:a)提出一种新的分布式频繁项集挖

收稿日期：2017-11-30；修回日期：2018-01-07　　基金项目：中南大学研究生科研创新项目（2017zzts612）

作者简介：郑静益（1993-），男，江苏泰兴人，硕士研究生，主要研究方向为数据挖掘、机器学习；邓晓衡（1974-），男（通信作者），湖南人，教授，博士，主要研究方向为无线网络通信、网格计算与分布式处理、边缘计算、大数据分析（dxh@csu.edu.cn）.

掘算法——IEBDA，利用项编码表示项及其在每条数据记录中的出现情况，利用编码按位相与的计算方式，避免重复扫描完整的数据集，大大减少计算时间。结合 Spark 分布式计算框架，设计新的并行化实现方法，将编码计算并行化的执行，在保证算法正确性的前提下，加快计算速度，进一步提高算法效率；b) 应用不同的数据集，验证算法在不同规模数据集上的效率，将该算法在大数据环境下与同类算法比较，证明其在多轮迭代下的优势，同时也说明该算法存在的缺陷并分析进一步优化的思路；

1 相关工作

本文研究 Apriori 及其变形算法在分布式上的改进，随着近几年大数据的引入，单机系统处理效率低的缺陷越发明显。于是，频繁项集挖掘研究的重心也转向了分布式计算环境。

Hadoop 是众多数据挖掘研究人员采用的分布式计算框架之一，尽管，Hadoop 将数据保存到 HDFS 并读回的方式，需要庞大的磁盘读写，并不适合迭代算法，许多基于 Hadoop 的 Apriori 改进算法还是被陆续提出并取得了一定的优化效果。Ye 等人[3]提出了 Apriori 算法在多机环境下的并行实现，实现了一个基于数据结构的 Apriori 算法的快速版本，并分析其性能。Lin 等人[4]提出了 SPC，FPC 和 DPC 三种基于 Hadoop 的 Map-Reduce 编程模型的 Apriori 变种算法，这三种算法在不同程度上契合 Hadoop 集群来进行挖掘，其中 DPC 利用该特征动态地组合各种长度的候选项，比单通道的 SPC 和固定通道组合的 FPC 表现要好。Li 等人[5]通过应用基本的 Map-Reduce 功能，迭代地产生频繁项目集，并实现了完全基于 Map-Reduce 的并行化 Apriori 算法。Yu 等人[6]提出了一种新的算法实现思路，在每个事务上生成可能的频繁项集，每次迭代中独立的挖掘频繁项集再合并结果，从而进一步提高运行效率。基于 Hadoop 的 Apriori 的其他改进版本可参看文献[7~11]。

以上所有的算法都是基于 Hadoop 框架，应用 Map-Reduce 编程模型实现，尽管这些算法都作出了不同程度的改进，Hadoop 的文件存储系统决定的大量磁盘读写操作却极大地限制了算法的性能。而近年来流行的 Spark 计算框架，与 Hadoop 基于磁盘的计算方式不同，可以有效地在内存中运行迭代式的算法。因此，许多研究人员将 Apriori 算法应用到 Spark 框架中，相较于各种基于 Hadoop 的 Apriori 改进算法，效率又有了质的提升。Qiu 等人[12]实现了名为 YAFIM 的基于 Spark 的 Apriori 算法，该算法充分利用 Spark 框架的特性，如 RDD 算子操作和基于内存的并行计算，算法简单而又高效，其计算比基于 Map-Reduce 的算法快了约 8 倍的速度。Yang 等人[13]引入了一种新的基于矩阵的修剪，以减少候选项集的数量，减小搜索开销。Sethi 等人[14]利用数据集的垂直布局来解决在每次迭代中扫描数据集的问题，并在不同的数据集上和 YAFIM 算法进行对比实验，验证算法性能。基于 Spark 的其他改进版本可参看文献[15-16]。

本文提出的 IEBDA 算法，采用项编码的方式，进一步优化算法的频繁项集计算和候选项集剪枝，在整个算法过程中，只需要扫描一次完整数据集生成频繁 1 项集即可开始算法的迭代，在迭代过程中无须再次扫描数据集，只需要对内存中的频繁 k 项集编码进行操作即可生成候选 k+1 项集。算法基于 Spark 集群实现，并行化地计算频繁项集，很好地适应当下的大数据趋势。

2 基于项编码的频繁项集挖掘算法

本章节将详细介绍基于项编码的频繁项集挖掘算法[17]。其中，2.1 节介绍编码规则，2.2 节则描述整个算法的挖掘迭代过程。

2.1 编码规则

基于项编码的频繁项集挖掘算法，是一种改进的使用垂直数据格式[18]挖掘频繁项集的算法，旨在减少对于整体数据的扫描次数和对候选项集进行剪枝，从而提高算法效率，由于采用编码的方式表示某个项在所有每条数据中的出现情况，因此支持度、置信度和提升度等指标[17]的判断将变得更为简便。

项编码的规则为：编码的长度为数据库中数据记录的条数，根据数据库中数据记录的排序，每一条记录对应编码中的一个位置，如果某个项出现在第 i 条数据记录中，就相应的将编码的一个位置设为‘1’，否则设为‘0’。例如，假设数据库中共有 5 条数据记录(T1，T2，…，T5)，而某一个项出现在数据记录 T2，T4 中，那么根据编码规则可以得到该项的编码为 01010。

2.2 迭代计算过程

应用上述方法，得到所有项的编码，每个项编码中‘1’的出现次数即为该项在整个数据集中的出现次数，若大于预先设置的最小支持度和数据总条数的乘积，即满足支持度条件，则该项属于频繁 1 项集，保存所有频繁 1 项集及其对应的编码。

对于频繁 k+1 项集的计算，则迭代地以频繁 k 项集为输入，将频繁 k 项集中的项编码两两进行按位与运算，例如 A 项的编码为 0001110110，B 项的编码为 0001011110，则{A，B}项集的编码为(0001110110)＆(0001011110)= 0001010110，该编码可反映 A、B 两项同时出现的情况，并根据‘1’出现的次数计算其是否满足最小支持度和最小置信度要求，如果满足要求，则项集{A，B}是一个频繁项集。得到所有满足条件的频繁 k+1 项集后，更新频繁项集，从而继续挖掘频繁 k+2 项集，以此类推，迭代直至频繁 k 项集为空时，算法停止。输出所有的频繁项集，从频繁 1 项集至频繁 k-1 项集。算法迭代过程算法 1 所示：

算法 1 基于项编码的频繁项集挖掘迭代过程.

输入：编码化的频繁 1 项集 *C*，最小支持度 *min_sup*，总记录数 *size*；

输出：频繁项集 *L*.

假设：sum(x)表示计算编码 x 中 ‘1’ 的出现次数

① C_k = C;

② while C_k.size > 0

```
③   Temp={};  /*临时变量，记录频繁 k+1 项集*/
④   for i in Cₖ
⑤     for j in Cₖ - i
⑥       if (i∪j).length - i.length == 1 and
sum(i&j) > size * min_sup  /*判断生成的项集是否是 k+1 项集，同时判断该项集是否满足支持度条件*/
⑦         L.add(i∪j);
Temp.add(i∪j);
⑧       end if
    end for
end for
⑨   Cₖ = Temp;  /*更新频繁 k 项集，用于下一轮迭代，挖掘下一轮的频繁 k+1 项集，整个迭代直到 Cₖ 为空停止*/
⑩ end while
```

3 IEBDA 算法

在本节中，本文详细介绍 IEBDA 算法，该算法基于 Spark 计算框架，借鉴基于项编码的频繁项集挖掘算法中，利用项编码表示每一项在每条数据中出现情况，并以利用这一特点进行剪枝的思想，致力于解决在每次迭代中重复扫描数据集的问题，从而减少 I/O 开销和节约内存空间，并行化地完成频繁项集的计算。3.1 节将介绍 Spark 平台，3.2 节则详细讲述该算法如何在 Spark 上实现。

3.1 Spark 平台

Spark[20]是由加州大学伯克利分校的 AMP 实验室所开源的一个通用的大规模数据快速处理引擎，其平台完全由 Scala 语言编写，专注于基于内存的分布式计算，由于计算的中间结果保存在内存中而不是像 Hadoop 一样保存在 HDFS 文件系统上，因此大大减少了 I/O 开销，在部分实验中，其性能甚至超过 Hadoop 的 Map-Reduce 编程模型上百倍[21]。

Spark 采用称为弹性分布式数据集（RDD）的数据抽象[20]，RDD 是一组不可变的数据对象，在实际的存储中数据保存在不同的集群节点上的。RDD 可以通过读取数据创建，也可以从其他 RDD 转换而成，它是一种只读的数据结构，一旦创建不可修改，从而保证数据的正确性。此外，Spark 使用磁盘来缓存数据计算中缺省的数据块，对于经常需要访问的数据，这种缓存方式可以实现对数据的快速访问，这一点对于迭代式的程序来说，可以大大减少运行时间，这也是本文选择 Spark 平台来实现算法改进的原因。Spark 以图的形式保存不同 RDD 之间的依赖关系信息，这种 RDD 关系图被称为谱系图[20]。任何 RDD 的丢失，都可以根据谱系图中的依赖关系，重新计算以恢复，从而保证整个 Spark 生态环境的容错性。Spark 的 RDD 编程支持多种编程语言，常用的例如 Java，Scala 和 Python 等，这一特点也使得其学习成本大幅度降低。

如图 1 所示是搭建在 HDFS 文件系统和 YARN 调度工具之上的 Spark 架构，在该架构中，Spark 集群中的主节点，接收来自用户的请求，通过 YARN 的 ResourceManager 进程将用户提交的应用程序分配到 Worker 节点上，分布式的运行各自的任务，数据一旦从 HDFS 文件系统中读入，就把处理的中间结果保存在缓存中，无须多次扫描磁盘获取数据，加快数据的访问速度。处理的结果可以写回到 HDFS 中也可以通过可视化的方式展示给用户。

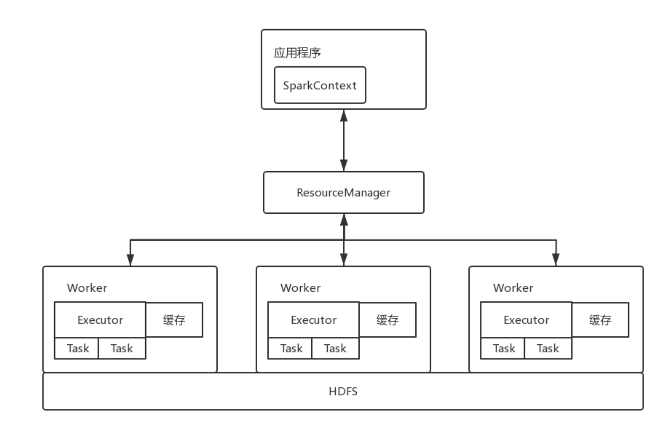


图 1 HDFS 文件系统和 YARN 调度工具之上的 Spark 架构图

3.2 IEBDA 算法的实现

在整个算法的执行中，大致分为两步，第一步扫描数据集生成频繁 1 项集，并把频繁 1 项集作为频繁 k 项集，存入 Spark 的广播变量（broadcast variable）。广播变量在整个 Spark 集群中为全部节点所共享，可以减少数据的访问的时间开销，因为频繁项集存储的形式是每个项集及其对应的编码的键值对形式，存储频繁项集占据的空间远小于存储完整数据集需要的空间。第二步，迭代的计算和输出频繁 k+1 项集，每次迭代，在每个节点上并行化地计算生成的 k+1 项集及其编码，并完成剪枝工作，剔除所有不满足支持度条件的 k+1 项集，即可得到频繁 k+1 项集，并把频繁 k+1 项集替换广播变量中的频繁 k 项集，开始下一轮的迭代，直至无法再产生满足支持度条件的更高级频繁项集。因为并行化地计算频繁 k+1 项集，在不同节点上，可能会产生重复的频繁项集，因此每轮迭代都需要去重，在小数据集上的表现可能较差，但是应用到大数据环境，去重所花费的时间相比于挖掘计算的时间影响不明显，因此算法的效率也能得到保证。下面将详细阐述 IEBDA 算法在 Spark 上的实现。

3.2.1 计算频繁 1 项集

计算频繁 1 项集流程如图 2 所示。

基于项编码的频繁项集挖掘算法处理的是使用垂直数据格式存储的数据，与之类似，IEBDA 算法输入的数据也需要先格式化。原始数据读取为成 RDD 结构后，通过 RDD 的 flatMap() 函数和 groupByKey()函数，得到所有项及其在每条数据记录中的出现情况，将原始的数据转换为形如<I_i, T_i>，其中 I_i 表示原始数据中的一个项，T_i 表示包含 I_i 的数据记录 id 的集合，即表示所有 I_i 出现的记录的集合。利用 RDD 的 map()函数，并行化的计算每一个项的编码，得到项编码的键值对数据<I_i , E_i>，其中 E_i 表示 I_i 所对应的编码。

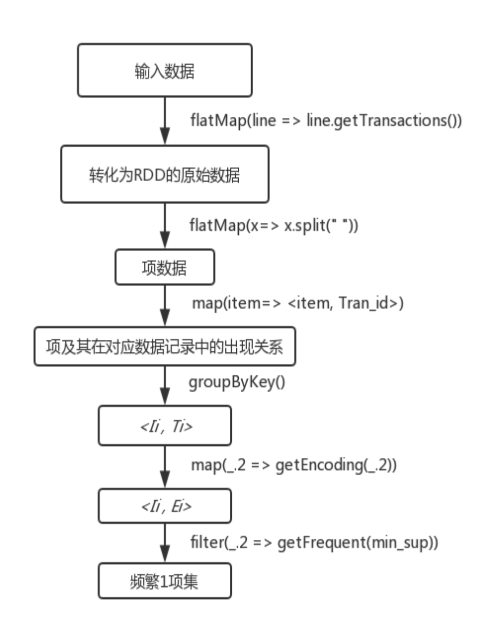


图 2　频繁 1 项集计算流程图

由此，就求出了所有 1 项集及其对应的编码数据，使用 RDD 的 filter()来去除不符合支持度要求的项编码键值对，留下的数据即为频繁 1 项集，将此数据设置为广播变量，写入到所有节点的缓存中，方便数据的访问。保存在广播变量中的频繁 1 项集数据将作为下一步迭代生成频繁 2 项集的频繁 k 项集，是整个迭代挖掘频繁 k+1 项集的开始，如果频繁 1 项集为空，则不会进行迭代，直接输出最后结果为空。如果频繁 1 项集不为空，则以此为迭代挖掘频繁 k 项集的输入数据，进行下一步的挖掘。这样的处理方式是为了减少后续迭代消耗的时间。

3.2.2 迭代挖掘频繁 k+1 项集

迭代挖掘频繁 k+1 项集的伪代码如算法 2 所示。

算法 2　IEBDA 算法的频繁项集挖掘迭代过程

```
输入：广播变量中的频繁 1 项集 B，最小支持度 min_sup，总记录数 count；
输出：频繁项集 L.
假设：sum(x)表示计算编码 x 中 ‘1’ 出现次数
① b = B.parallelize();  /*并行化*/
L.addAll(B)
② while b.size > 0
③   Temp={};  /*临时变量，记录频繁 k+1 项集*/
④   foreach i in b
⑤     for j in B
⑥        if (i._1∪j._1).length - i._1.length == 1 and sum(i._2&j._2) > count * min_sup  /*判断生成的项集是否是 k+1 项集，同时判断该项集是否满足支持度条件*/
⑦          L.add(i∪j);
Temp.add(i∪j);
⑧        end if
end for
end for
⑨   Temp.dropDumplicates();  /*分布式下需要进行汇总去重*/
b = Temp;
B = broadCast(Temp);
/*更新频繁 k 项集和广播变量，用于下一轮迭代，挖掘下一轮的频繁 k+1 项集，整个迭代直到 b 为空停止*/
⑩ end while
```

在每一次的迭代中，都将保存在广播变量中的频繁 k 项集读取为 RDD 数据结构，利用 map()函数对该 RDD 函数中的每一个项集与广播变量中的候选项集求并集，这一步是并行计算，因此比单机上的循环要高效很多。根据 Apriori 算法的原理，本文从频繁 k 项集生成候选 k+1 项集，在这里得到的项集的长度必须比原来的项集的长度大 1。

过滤不符合条件的项集后，计算所有候选 k+1 项集的编码，根据 RDD 的计算方式，这一步也可以并行化地处理，但是本步骤并行化处理会生成部分重复的项集，在分布式计算时，为了提高效率，先保留所有项集，在每轮迭代结束前再收集各个节点的结果，统一进行去重操作。编码的计算，只需要将做并运算得到该候选 k+1 项集的两个 k 项集的编码做按位与运算即可。得到所有候选 k+1 项集及其编码后，再利用和判断频繁 1 项集一样的方法，来判断候选 k+1 项集是否符合频繁项集的条件，从而得到所有的频繁 k+1 项集。收集所有节点的结果并完成去重。

若生成的频繁 k+1 项集不为空，则将其作为新的广播变量，替换掉原来的频繁 k 项集，进而迭代计算，直到频繁 k+1 项集为空，输出 2.1 节中得到的频繁 1 项集和本节中迭代计算出的每一轮的频繁 k 项集，作为最终的结果。

3.2.3 总体流程

在整个算法中，第一步的挖掘过程将原数据集转换成编码格式，并作为广播变量分发到所有节点，这一步相较已有的算法，消耗时间会更长，但是在后续迭代过程中，不需要再扫描完整数据集，只需要在频繁 1 项集编码的基础上进行计算即可，因此，随着迭代轮数的增加，该算法的整体优势将会体现出来，在本文 4.1 节中将着重分析该算法在大数据多轮迭代下的优势。

如图 3 所示即为 IEBDA 算法的整体流程图。

4　实验与结果

本章将 IEBDA 算法和其他频繁项集挖掘算法应用到如表 1 所示的数据集上。其中，校园一卡通数据集来自中南大学数据云平台提供的学生选修课数据，实验中选取软件学院 3 个年级共 2 842 名学生的选课数据，学院提供的选修课共计 27 门。蘑菇数据集是加州大学欧文分校 UCI 数据库提供的机器学习开源数据集，实验中以去掉全部空值属性后剩下的属性值为项进行实验，原始数据的下载地址为 http://archive.ics.uci.edu/ml/datasets/Mushroom，Retail 数据集为 FIMI(Frequent Itemset Mining Implementations Repository)开源的数据集，下载地址为 http://fimi.ua.ac.be/data/retail.dat。

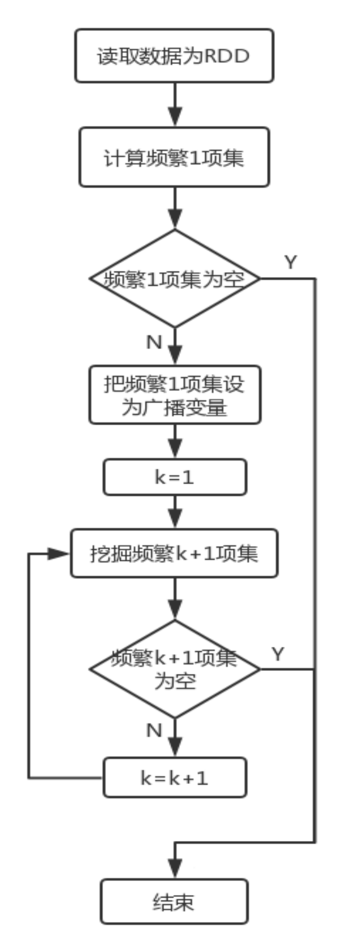


图 3　IEBDA 算法流程图

表 1　数据集

数据集	项数	数据记录条数
校园一卡通数据集	27	2842
蘑菇数据集	122	8416
Retail 数据集	16469	87988

如表 2 所示则是在 2 核 cpu，8G 内存，1T 硬盘存储空间单机环境下，采用基于项编码的频繁项集挖掘算法，挖掘表 1 中的三个数据集所需时间。在实验中，对各数据集，设置最小支持度为 0.35，运行 5 次算法的代码，取这 5 次运行时间的平均值作为各个数据集上运行基于项编码的频繁项集挖掘算法所需的时间。由表中的数据可知，单机版的基于项编码的频繁项集挖掘算法在小数据集上的表现非常好，但随着数据量的增加，算法效率急剧下降，而在 Retail 这样的大数据集上，由于内存等限制，算法将无法运行出最终结果。因此，在大数据环境下，分布式的 Apriori 算法是有必要的。

表 2　基于项编码的频繁项集挖掘算法在数据集上的运行时间

数据集	平均时间/s
校园一卡通数据集	2.147
蘑菇数据集	871.562
Retail 数据集	—

4.1　并行化算法效率测试

实验集群环境如下：搭建以 1 台双核 cpu，4G 内存，1T 硬盘存储空间的电脑作为主节点；两台 6 核 cpu，64G 内存，12T 硬盘存储空间为从节点的集群，jdk 版本为 1.7，scala 版本为 2.10.4，Hadoop 版本为 2.5.2，Spark 版本为 2.0.2。实验中本文主要把 IEBDA 算法和表现较好的基于 Hadoop 的分布式 DPC 算法[4]以及基于 Spark 的 YAFIM 算法[12]和 HFIM 算法[14]进行比较，证明算法的性能。

如图 4 所示，是这几个算法在每一轮迭代中所消耗时间的对比，为了防止算法运行迭代的次数过少，本文给每个数据集上设置了各自的最小支持度，使得迭代的次数在 7 次左右。图 4 中子图 a 为校园一卡通数据集上设置最小支持度为 0.35 的运行时间对比，子图 b 为蘑菇数据集上设置最小支持度为 0.35 的算法时间运行对比，子图 c 为 Retail 数据集上设置最小支持度为 0.075 的算法运行时间对比。整体上看，基于 Spark 的三个算法要比基于 Hadoop 的 DPC 算法效率高上许多，Spark 基于内存的计算相比于 Hadoop 优势明显。

如图 4 子图 a 可知，各分布式算法，在校园一卡通数据集上的总运行时间，都高于表 2 中所示同样条件下单机版基于项编码的频繁项集挖掘算法所需的 2.147 秒，因为在单机能处理的数据量下，网络通信的开销相对影响更大，基于单机内存的计算效率也因此远高于通过网络连接的分布式的计算效率。因此，基于分布式的频繁项集挖掘算法并不适用于小数据集的场景。相反，在子图 b 和子图 c 中，原先单机版算法挖掘效率较低甚至无法完成的挖掘的任务，都可以在可接受的时间内完成，所以，在数据量较大的场景之下，通过分布式的手段，能有效保证算法的效率。

如图 4 三张子图的第一轮迭代情况可知，由于 IEBDA 算法在迭代开始之前，会把数据集转换为频繁 1 项集的编码，并将其加入到广播变量，因此 IEBDA 算法在第一轮迭代时，消耗时间往往会比 YAFIM 算法和 HFIM 算法消耗时间要长，但是在后续的迭代过程中，广播变量加快了数据的访问速度，因而在除了第一轮迭代之后的几轮迭代中，IEBDA 算法运行速度明显提升，超过其他两种算法，这一点使得在多轮迭代的情况下，该算法运行总时间少于其他分布式算法。可想而知，如果迭代轮数较少，则第一轮迭代多花费的时间不一定能被后续几轮的加速弥补，因此 IEBDA 算法最适合的场景还是大数据环境下的多轮迭代场景。

如图 4 子图 a 中第六轮迭代可知，当迭代到了最后阶段，频繁项集可能已经非常少，此时 IEBDA 算法中通过广播变量计算频繁项集的方式在速度上不如 YAFIM 算法中基于树结构的查询和 HFIM 垂直结构的数据查询，因此在最后一轮迭代时 YAFIM 算法和 HFIM 算法的效率可能会高于 IEBDA 算法，但是由于频繁项集过少，这种时间差几乎可以忽略不计，因此，整体上来说 IEBDA 算法在多轮迭代的条件下，算法性能还是优于 YAFIM 算法和 HFIM 算法，也因而更适合于迭代挖掘次

数较多的情况。

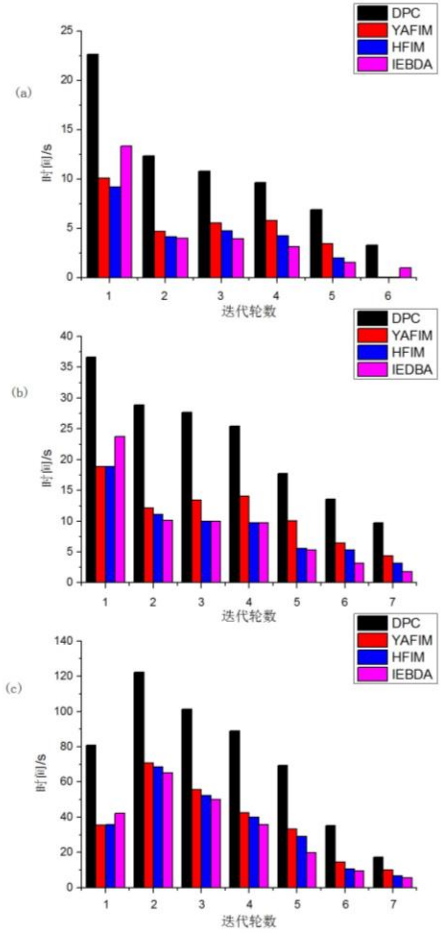


图 4　分布式算法每轮迭代消耗时间

4.2　集群扩展性测试

IEBDA 算法适用于大数据环境下多轮迭代场景。因此本节的实验使用 Retail 数据集作为实验数据集，设置最小支持度为 0.075。实验利用虚拟机模拟双核 cpu、6G 内存、1T 存储空间的节点，分别测试节点数从 1~5 的集群条件下，记录运行十次完整的 IEBDA 算法所花费的平均时间。

如图 5 所示，在 3 个节点以下，节点数与运行时间大致成线性关系；当节点数大于 3 时，随着节点的增加，算法效率的提升速度减缓，整体上来看，集群节点越多，计算能力越强，挖掘所用的总时间也越短，所以，算法效率随着节点数的增加而增加。

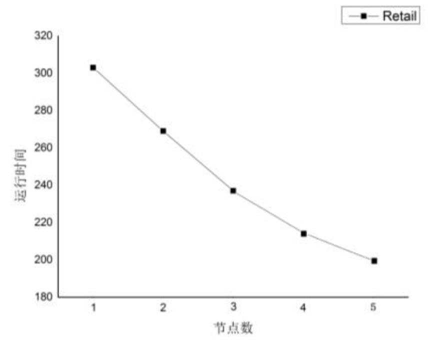


图 5　IEDBA 算法在不同节点集群下的运行速度提升

4.3　算法优缺点分析

综上所述，本文提出的 IEBDA 算法，充分利用项编码表达数据信息，省去了耗时的数据集重复扫描，简化了计算。基于 Spark 平台实现的并行化处理，进一步提高算法的效率。应对于目前流行的大数据环境，在迭代挖掘次数较多的场景下，该算法具有比 YAFIM 算法和 HFIM 算法更高的效率。

但是该算法仍有值得进一步改进的地方。第一点，由于该算法是基于传统的 Apriori 算法进行改进，候选 k+1 项集由频繁 k 项集生成的规则并没有改变，每一步迭代需要更新频繁 k 项集进行下一轮的挖掘，虽然避免重复扫描数据集，但是更新操作在分布式环境下的通信开销仍然是不可忽视的，所以，一个可能的解决思路是利用已有的频繁 k 项集生成候选 k+n 项集而不仅仅是候选 k+1 项集。第二点，随着迭代的进行，如果频繁 k 项集的数量已经减小到了单机能处理的范围内，那么继续使用 Spark 的广播变量，效率不见得比单机计算高，还会增加额外的网络通信开销，传统的设置阈值判断使用分布式处理还是单机处理的方法在这种场景下不能解决这一问题，因为这种方式会在每一轮迭代时判断分散存储在不同节点上的数据的总容量是否小于阈值，而这样的处理本身也会增加通信开销，另外，出现频繁 k 项集的数量能被单机处理的情况在实际应用中所占比重也很小，这意味着迭代过程中做的多数判断实际上属于“无用功”，因此这种方法不适用。目前看来，由于这种情况在大数据环境下本身出现得比较少，即使出现导致的计算效率下降很小，因此忽略不计保持整体计算速度似乎是比较好的策略。

5　结束语

虽然本文提出的算法在大数据环境下，比原先的分布式算法更适合于迭代挖掘次数较多的场景，但是还有很多值得进一步研究的问题。在频繁项集的生成规则上，可以考虑改进原先的迭代式生成规则，充分利用已有的频繁 1 项集；在分布式环境下，如何采用合适的数据结构减少通信开销，进一步加快算法计算速度，也是值得探索的问题。因此，将继续研究频繁项集挖掘算法的改进。

参考文献：

[1] Jiao Y. Research of an improved Apriori algorithm in data mining association rules [J]. International Journal of Computer & Communication Engineering, 2013, 2 (1): 25-27.

[2] Borgelt C, Kruse R. Induction of association rules: Apriori implementation [M]// Compstat. [S. l.] : Physica-Verlag HD, 2002: 937-944.

[3] Ye Y, Chiang C C. A parallel Apriori algorithm for frequent itemsets mining [C]// Proc of International Conference on Software Engineering Research, Management and Applications. Washington DC: IEEE Computer Society, 2006: 87-94.

[4] Lin X. MR-Apriori: association rules algorithm based on MapReduce [C]//

Proc of IEEE International Conference on Software Engineering and Service Science. 2014: 141-144.

[5] Li N, Zeng L, He Q, *et al.* Parallel implementation of Apriori algorithm based on MapReduce [C]// Proc of ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing. 2012: 236-241.

[6] Yu R M, Lee M G, Huang Y S, *et al.* An efficient frequent patterns mining algorithm based on MapReduce framework [C]// Proc of International Conference on Software Intelligence Technologies and Applications & International Conference on Frontiers of Internet of Things. 2015: 1-5.

[7] Yang X Y, Liu Z, Fu Y. MapReduce as a programming model for association rules algorithm on Hadoop [C]// Proc of International Conference on Information Sciences and Interaction Sciences. 2010: 99-102.

[8] Dharavath R, Kumar V, Kumar C, *et al.* An Apriori-based vertical fragmentation technique for heterogeneous distributed database transactions [M]// Intelligent Computing, Networking, and Informatics. [S. l.] : Springer, 2014: 687-695.

[9] 田卫东，许静文. 基于模糊等价类的频繁项集精简表示方法 [J]. 计算机应用研究, 2016, 33 (7): 1936-1940.

[10] 唐颖峰，陈世平. 一种基于后缀项表的并行闭频繁项集挖掘算 法 [J]. 计算机应用研究, 2014, 31 (2): 373-377.

[11] 谢志明，王鹏. 基于 MapReduce 架构的并行矩阵 Apriori 算法 [J]. 计算机应用研究, 2017, 34 (2): 401-404.

[12] Qiu H, Gu R, Yuan C, *et al.* YAFIM: a parallel frequent itemset mining algorithm with Spark [C]// Proc of IEEE International Parallel & Distributed Processing Symposium Workshops. Washington DC: IEEE Computer Society, 2014: 1664-1671.

[13] Yang S, Xu G, Wang Z, *et al.* The parallel improved apriori algorithm research based on Spark [C]// Proc of the 9th International Conference on Frontier of Computer Science and Technology. 2015: 354-359.

[14] Sethi K K, Ramesh D. HFIM: a Spark-based hybrid frequent itemset mining algorithm for big data processing [J]. Journal of Supercomputing, 2017, 73 (8): 3652–3668.

[15] Rathee S, Kaul M, Kashyap A. R-Apriori: an efficient apriori based algorithm on Spark [C]// Proc of Workshop in Information and Knowledge Management. New York: ACM Press, 2015: 27-34.

[16] Gui F, Ma Y, Zhang F, *et al.* A distributed frequent itemset mining algorithm based on Spark [C]// Proc of International Conference on Computer Supported Cooperative Work in Design. 2015: 271-275.

[17] Pang S, Kasabov N. Encoding and decoding the knowledge of association rules over SVM classification trees [J]. Knowledge & Information Systems, 2009, 19 (1): 79-105.

[18] 邢长征，安维国，王星. 垂直数据格式挖掘频繁项集算法的改进 [J]. 计算机工程与科学, 2017, 39 (7): 1365-1370.

[19] Pudi V. Data mining: concepts and techniques [M]. Oxford: Oxford University Press, 2009.

[20] Bell J. Apache Spark [M]// Machine Learning: Hands-on for Developers and Technical Professionals. Hokebon: Wiley, 2015: 275-314.

[21] Zaharia M, Chowdhury M, Das T, *et al.* Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing [C]// Proc of USENIX Conference on Networked Systems Design and Implementation. [S. l.] : USENIX Association, 2012.